On the Complexity of Exact Pattern Matching in Graphs: Binary Strings and Bounded Degree

Massimo Equi, Roberto Grossi, Veli Makinen

On the Complexity of Exact Pattern Matching in Graphs: Binary Strings and Bounded Degree

Massimo Equi
Department of Computer Science, University of Helsinki, Finland
massimo.equi@helsinki.fi

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Italy
grossi@di.unipi.it

Veli Mäkinen
Department of Computer Science, University of Helsinki, Finland
veli.makinen@helsinki.fi

Abstract

Exact pattern matching in labeled graphs is the problem of searching paths of a graph $G = (V, E)$ that spell the same string as the pattern $P[1..m]$. This basic problem can be found at the heart of more complex operations on variation graphs in computational biology, of query operations in graph databases, and of analysis operations in heterogeneous networks, where the nodes of some paths must match a sequence of labels or types. We describe a simple conditional lower bound that, for any constant $\epsilon > 0$, an $O(|E|^{1-\epsilon}\, m)$-time or an $O(|E|\, m^{1-\epsilon})$-time algorithm for exact pattern matching on graphs, with node labels and patterns drawn from a binary alphabet, cannot be achieved unless the Strong Exponential Time Hypothesis (SETH) is false. The result holds even if restricted to undirected graphs of maximum degree three or directed acyclic graphs of maximum sum of indegree and outdegree three. Although a conditional lower bound of this kind can be somehow derived from previous results (Backurs and Indyk, FOCS'16), we give a direct reduction from SETH for dissemination purposes, as the result might interest researchers from several areas, such as computational biology, graph database, and graph mining, as mentioned before. Indeed, as approximate pattern matching on graphs can be solved in $O(|E|\, m)$ time, exact and approximate matching are thus equally hard (quadratic time) on graphs under the SETH assumption. In comparison, the same problems restricted to strings have linear time vs quadratic time solutions, respectively, where the latter ones have a matching SETH lower bound on computing the edit distance of two strings (Backurs and Indyk, STOC'15).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms Theory of computation → Problems, reductions and completeness Theory of computation → Pattern matching

Keywords and phrases exact pattern matching, graph query, graph search, heterogeneous networks, labeled graphs, string matching, string search, strong exponential time hypothesis, variation graphs

Funding This work has been partially supported by Academy of Finland (grant 309048)

1 Introduction

Large-scale labeled graphs are becoming ubiquitous in several areas, such as computational biology, graph databases, and graph mining. Applications require sophisticated operations on these graphs, and often rely on primitives that locate paths whose nodes have labels or types matching a pattern given at query time.

In graph databases, query languages provide the user with the ability to select paths based on the labels of their nodes or edges, where the edge labels are called properties. In this way, graph databases explicitly lay out the dependencies between the nodes of data, whereas these dependencies are implicit in classical relational databases [4]. Although a standard query language has not been yet universally adopted (as it occurred for SQL in relational databases), popular query languages such as Cypher [13], Gremlin [25], and SPARQL [23] offer the possibility of specifying paths by matching the labels of their nodes.

In graph mining and machine learning for network analysis, heterogeneous networks specify the type of each node [27]. For example, in the DBLP network [30], the nodes for authors can be marked with letter 'A', and the nodes for papers can be marked with letter 'P', where edges connect authors to their papers. For example, coauthors can be identified by the pattern 'APA' when it matches two different nodes with 'A'. The strings generated by the labels on the paths have several applications in heterogeneous networks, such as graph kernels [16] or node similarity [10], where a basic tool is retrieving the paths for a string.

In genome research, the very first step of many standard analysis pipelines of high-throughput sequencing data has been to align the sequenced fragments of DNA (called reads) on a reference genome of a species. Further analysis reveals a set of positions where the sequenced individual differs from the reference genome. After years of these kind of studies, there is now a growing dataset of frequently observed differences between individuals and the reference. A natural representation of this gained knowledge is a *variation graph* where the reference sequence is the backbone and variations are encoded as alternative paths [26]. Aligning reads (pattern matching) on this labeled graph gives the basis for the new paradigm called *computational pan-genomics* [9]. There are already tools that use such ideas, e.g. [15].

Although there is a growing need to perform pattern matching on graphs in several situations described above, the idea of extending the problem of string searching in sequences to pattern matching in graphs was studied over 25 years ago as a search problem in *hypertext* [20]. The history of key contributions is given in Table 1, where the two best known results for exact and approximate pattern matching, both taking quadratic time in the worst case, are highlighted. Note that errors in the graphs makes the problem NP-hard [3], so we consider errors in the pattern only.

A common feature of the bounds reported in Table 1 is the appearance of the quadratic term $m\,|E|$ (except for the special cases of trees and the general NP-hard approximate version). Here m is the length of the pattern string and E is the set of edges of the graph. The quadratic cost of the approximate matching on graphs by Rautiainen and Marschall [24] are asymptotical optimal under the Strong Exponential Time Hypothesis [17] (SETH) as (i) they solve the approximate string matching as a special case, since a graph consisting of just one path of $|E|+1$ nodes and $|E|$ edges is a text string of length $n = |E|+1$, and (ii) it has been recently proved that the edit distance of two strings of length n cannot be computed in $O(n^{2-\epsilon})$ time, for any constant $\epsilon > 0$, unless SETH is false [5]. Hence this conditional lower bound explains why the $O(m|E|)$ barrier has been difficult to cross.

We can only explain the complexity on *approximate* pattern matching on graphs, but nothing is known on *exact* pattern matching on graphs. Indeed, the classical exact pattern matching with a pattern and a text string can be solved in linear time [19], so one could expect the corresponding problem on graphs to be easier than approximate pattern matching.

In this paper we end up with a slightly surprising observation that *exact and approximate pattern matching are equally hard on graphs*. Namely, we show the conditional lower bound that an $O(|E|^{1-\epsilon}\,m)$-time or an $O(|E|\,m^{1-\epsilon})$-time algorithm for exact pattern matching on graphs cannot be achieved unless SETH is false. This result explains why it has been difficult

State of the art for *PMLG*				
Year	**Authors**	**Graph**	**Exact/ Approximate**	**Time**
1992	Manber, Wu [20]	DAG	approximate$^{(1)}$	$O(m\|E\| + occ \lg\lg m)$
1993	Akutsu [2]	Tree	exact	$O(N)$
1995	Park, Kim [22]	DAG	exact$^{(3)}$	$O(N + m\|E\|)$
1997	Amir et al. [3]	general	exact	$O(N + m\|E\|)$
1997	Amir et al. [3]	general	approximate$^{(2)}$	NP-Hard
1997	Amir et al. [3]	general	approximate$^{(1)}$	$O(Nm \lg N + m\|E\|)$
1998	Navarro [21]	general	approximate$^{(1)}$	$O(Nm + m\|E\|)$
2017	Vadaddi et al. [29]	general	approximate$^{(1)}$	$O((\|V\| + 1)m\|E\|)$
2017	Rautiainen, Marschall [24]	general	approximate$^{(1)}$	$O(N + m\|E\|)$
2019	Jain et al. [18]	general	approximate$^{(2)}$	NP-Hard on binary alphabet

Table 1 Legend: V = set of nodes, E = set of edges, occ = number of matches for the pattern in the graph, m = pattern length, N = total length of text in all nodes, *(1)* errors only in the pattern, *(2)* errors in the graph, *(3)* matches span only one edge. The two rows highlighted in gray report the best known bounds for exact and approximate pattern matching.

to find indexing schemes for graphs with other than best case or average case guarantees for fast exact pattern matching [28, 14].

Before going on to give the overview and details of the reduction, let us now fix the problem definition and SETH formulation.

▶ **Definition 1.** Given an alphabet Σ, a labeled graph G is a triplet (V, E, L) where (V, E) is a directed or undirected graph and $L : V \to \Sigma^+$ is a function that defines which string over Σ is assigned to each node.

▶ **Definition 2.** Let $u_1, \ldots, u_j$ be a path in graph G and P be a pattern. Also, $L(u)[l :]$ and $L(u)[: l']$ denote the suffix of $L(u)$ starting at position l and the prefix of $L(u)$ ending at position l', respectively. We say that $u_1, \ldots, u_j$ is a match for P in G with offset l if the concatenation of the strings $L(u_1)[l :] \cdot L(u_2) \cdot \ldots \cdot L(u_{j-1}) \cdot L(u_j)[: l']$ equals P, for some l'.

The *Pattern Matching in Labeled Graphs* (PMLG) problem is then defined as:
INPUT: a labeled graph $G = (V, E, L)$ and a pattern P over an alphabet Σ.
OUTPUT: all the matches for P in G.

For example, in Fig. 1 pattern `cde` has two occurrences but pattern `bccce` does not occur. For our purpose it would be enough to exploit a decision version of the problem, namely, to be able to determine whether or not there exists at least one match for P in G, without reporting all of them. Note that the matching path for P can go through the same nodes multiples times in G as otherwise PMLG is trying to solve the NP-hard Hamiltonian path problem.

We now recall what is SETH, namely, the Strong Exponential Time Hypothesis [17]. This is a conjecture which is commonly used as a basis of reductions in the scientific community, even though its weaker version ETH is more widely accepted.

▶ **Definition 3** ([17])**.** Let q-SAT be an instance of SAT with at most q literals per clause. Given $\delta_q = \inf\{\alpha : \text{there is an } O(2^{\alpha n})\text{-time algorithm for } q\text{-SAT}\}$, SETH claims that $\lim_{q\to\infty} \delta_q = 1$.

In other words, it is hard to find an $O(2^{\alpha n})$-time algorithm for general SAT for a constant $\alpha < 1$. We use SETH in the the following result, given that the best known algorithm for PMLG, devised 20 years ago [3], has an $O(|E|\, m)$ time complexity.

▶ **Theorem 4.** *For any constant $\epsilon > 0$, the Pattern Matching in Labeled Graphs (PMLG) problem for an alphabet of at least 4 symbols cannot be solved in either $O(|E|^{1-\epsilon}\, m)$ or $O(|E|\, m^{1-\epsilon})$ time unless SETH is false.*

We can further strengthen the statement of this theorem by proving the following corollaries.

▶ **Corollary 5.** *The conditional lower bound stated in Theorem 4 holds even if it is restricted to graphs with binary alphabet for the labels, where each node has degree at most three.*

▶ **Corollary 6.** *The conditional lower bound stated in Theorem 4 holds even if it is restricted to labeled directed acyclic graphs (DAGs) with binary alphabet for the labels, where each node has the sum of indegree and outdegree at most three.*

In order to achieve these results we break down our reasoning process in some intermediate steps. Since this is a conditional lower bound we will reduce SAT to PMLG. Then we will show that having a truly subquadratic algorithm for PMLG would cause to solve SAT in $O(2^{\alpha n})$ time with $\alpha < 1$. Our reduction costs $\tilde{O}(2^{\frac{n}{2}})$ time for a SAT formula with n variables and $k = O(\text{poly}(n))$ clauses, where $\tilde{O}$ is the shorthand for ignoring polynomial factors in n, e.g. $O(kn^2\, 2^{\frac{n}{2}}) = \tilde{O}(2^{\frac{n}{2}})$. Hence the main steps can be synthesized as follows.

- Find a reduction from SAT to PMLG.
- Ensure that this reduction costs $\tilde{O}(2^{\frac{n}{2}})$ time.
- Show that having a $O(|E|^{1-\epsilon}\, m)$ or a $O(|E|\, m^{1-\epsilon})$ time algorithm for PMLG gives a solution for SAT that makes SETH fail.

Our reduction shares some similarities with those for string problems in [5, 8, 1, 6, 7] as it uses SETH. The closest connection is with [6], where regular expression matching is studied (graph G_F in Section 2.2 is analogous to the NFA derived from the regular expression matching of type $|\cdot|$ in [6]). At presentation level, the difference to earlier work is that we reduce directly from SETH, while the earlier work uses an intermediate problem, orthogonal vectors, as a tool; our reduction can also be presented via the orthogonal vectors problem, but we preferred to work with SETH directly since SAT is more familiar to researcher from various research areas. On a more conceptual level, the new reduction has some interesting features of independent interest. Given a SAT formula, our reduction builds a pattern and a graph, using some special characters in the pattern to match bridges in the graph that can be traversed in one direction only (even if the graph is undirected). Also, obtaining the reduction for a binary alphabet requires a suitable variable-length encoding of the characters to avoid certain paths in the graph.

An earlier version of this reduction can be found in the Master's thesis of the first author [11] (supervised by the two last authors).

2 Conditional lower bound for PMLG on undirected graphs

Consider a SAT formula F with variables $v_1, \ldots, v_n$ and set C of k clauses.[1] We show how to generate a corresponding instance of PMLG. We build a pattern $P \in \Sigma^m$ of suitable length

[1] In this paper we discuss the interesting case where $k = O(\text{poly}(n))$.

$m = \tilde{O}(2^{\frac{n}{2}})$ and a labeled graph $G = (V, E, L)$, where $|E| = \tilde{O}(2^{\frac{n}{2}})$ and $L : V \to \Sigma^*$ is the node labeling with strings from Σ^*, such that P matches in G if and only if F is satisfied by some truth assignment of its variables. Recall that a truth assignment x is a tuple $\langle b_1, \ldots, b_n \rangle$, where $b_i \in \{\texttt{true}, \texttt{false}\}$ is the truth value assigned to each variable v_i. We write $x \models c$ to indicate that there exists at least one literal $\ell \in c$ satisfied by x (i.e. either $\ell = v_i$ and $b_i = \texttt{true}$, or $\ell = \neg v_i$ and $b_i = \texttt{false}$).

Our reduction builds a pattern with $m = \tilde{O}(2^{\frac{n}{2}})$ symbols from a binary alphabet Σ along with an undirected graph whose nodes are labeled with single symbols from Σ (i.e. $L : V \to \Sigma$). This graph has $|V|, |E| = \tilde{O}(2^{\frac{n}{2}})$ nodes and edges, and maximum degree three. The reduction can be modified so that the graph is directed with maximum sum of indegree and outdegree at least three.

For presentation's sake, we begin with a pattern P using an alphabet of four symbols, $\Sigma = \{\texttt{b}, \texttt{e}, \texttt{c}, \texttt{d}\}$, whose interpretation is to label nodes according to their implicit functionality: **b**egin (synchronization token), **e**nd (synchronization token), **c**lause (marker), **d**ummy (don't care); moreover, the resulting undirected graph G has unbounded degree; after that, we will show how to get the minimal degree configuration for G and how to achieve a binary alphabet, as depicted above.

We assume that n is an even number, without loss of generality, and denote by X the set of $2^{\frac{n}{2}}$ possible assignments for the first $n/2$ variables, and by Y those for the last $n/2$ variables, that is,

$$X = \{x_i \mid x_i = \langle b_1^{(i)}, \ldots, b_{\frac{n}{2}}^{(i)} \rangle \text{ is a truth assignment for } v_1, \ldots, v_{\frac{n}{2}}\} \text{ and}$$

$$Y = \{y_j \mid y_j = \langle b_{\frac{n}{2}+1}^{(j)}, \ldots, b_n^{(j)} \rangle \text{ is a truth assignment for } v_{\frac{n}{2}+1}, \ldots, v_n\}.$$

We call elements of X and Y *half-assignments* and interpret notation $\models$ accordingly. For example, $y_j \models c$ if and only if there is a literal $\ell \in c$ satisfied by the half-assignment y_j (i.e. either $\ell = v_i$ and $b_i^{(j)} = \texttt{true}$, or $\ell = \neg v_i$ and $b_i^{(j)} = \texttt{false}$, for some $i \geq \frac{n}{2} + 1$).

The reduction components to follow will be interpreted as follows. The pattern encodes by position, placing a symbol **c** to indicate which clauses *cannot be satisfied* by a half-assignment $x_i \in X$; the other clauses are marked by **d** as they are already satisfied by x_i alone; symbols **b** and **e** are employed to sync the half-assignments from X with portions of the graph, called *gadgets*.

The gadgets encode which clauses *are satisfied* by the half-assignments of Y, encoding each such clause with a distinct node labeled with **c**: when a symbol **c** in the pattern matches a node with label **c** in the graph, the corresponding clause is now covered by a half-assignment $y_j \in Y$, while it was not yet covered by half-assignment $x_i \in X$. If all the symbols **c** for x_i are matched by the nodes of the gadget corresponding to y_j, then assignment $x_i y_j$ satisfyes the SAT formula F; also the other direction holds.

Parallel nodes labeled with **d** are introduced to deal with the cases when the pattern indicates that the corresponding clause is already satisfied by a half-assignment in X. Nodes labeled with **b** or **e** are used to match a half-assignment $x_i \in X$ with a half-assignment of $y_j \in Y$. Details follow below.

2.1 Building the pattern

Pattern P is defined over the alphabet $\Sigma = \{\texttt{b}, \texttt{e}, \texttt{c}, \texttt{d}\}$ using the half-assignments in $X = \{x_1, \ldots, x_{2^{\frac{n}{2}}}\}$ and the set $C = \{c_1, \ldots, c_k\}$ of clauses of SAT formula F. Specifically, it is built as the concatenation $P = \texttt{eb}P_{x_1}\texttt{e}\,\texttt{b}P_{x_2}\texttt{e}\ldots\texttt{b}P_{x_{2^{\frac{n}{2}}}}\texttt{eb}$ of $2^{\frac{n}{2}}$ strings where $x_i \in X$ and,

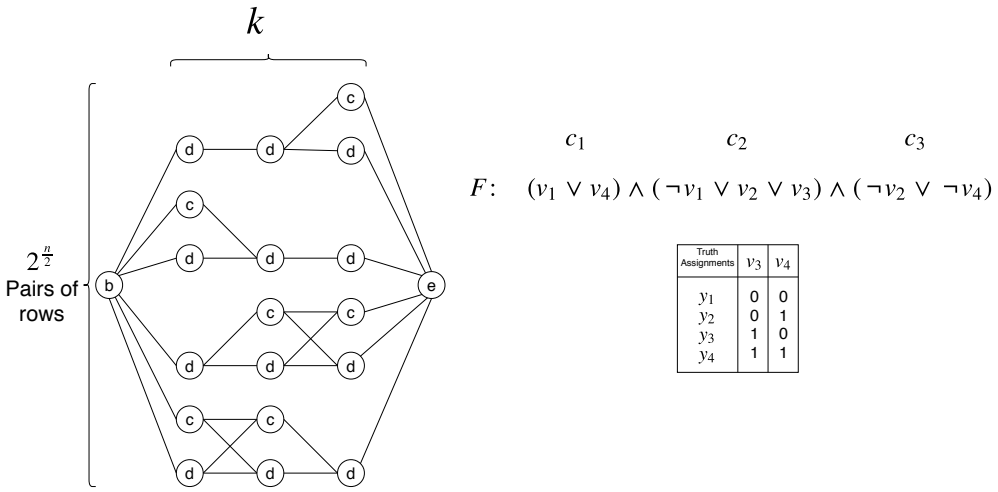

Figure 1 Gadget G_F for formula F, where $n = 4, 2^{\frac{n}{2}} = 4$ and $k = 3$. If for instance we look at the first column we observe that $c_{1,1}$ and $c_{3,1}$ are missing meaning that $y_1 \not\models c_1$ and $y_3 \not\models c_1$. On the other hand we know that $y_2 \models c_1$ and $y_4 \models c_1$ since we have nodes $c_{2,1}$ and $c_{4,1}$. An example of a pattern that we are able to match is $P = \texttt{bccde}$ while we would fail on $\bar{P} = \texttt{bcdce}$.

for $1 \leq h \leq k$, the hth symbol of string P_{x_i} is defined as

$$P_{x_i}[h] = \begin{cases} \texttt{c} & \text{if} \quad x_i \not\models c_h \\ \texttt{d} & \text{otherwise} \end{cases}$$

We will prove that F is satisfiable if and only if we can find a match for this pattern in our graph, where the latter is made up of gadgets as specified below.

2.2 Graph gadgets for SAT formulas

Our gadget is an undirected graph $G_F = (V_F, E_F, L_F)$, illustrated in Figure 1 and defined as follows using the $2^{\frac{n}{2}}$ half-assignments in $Y = \{y_1, \ldots, y_{2^{\frac{n}{2}}}\}$ and the set $C = \{c_1, \ldots, c_k\}$ of clauses of SAT formula F.

In the set V_F of nodes, we have a clause node $c_{j,h}$ for every possible pair $y_j, c_h \in Y \times C$ such that $y_j \models c_h$, and a dummy node $d_{j,h}$ for every possible pair $y_j, c_h \in Y \times C$. Set V_F also contains two special nodes, a begin node b and an end node e,

$$V_F = \{c_{j,h} \mid y_j \models c_h,\ y_j \in Y, c_h \in C\} \cup \{d_{j,h} \mid y_j \in Y, c_h \in C\} \cup \{b, e\}$$

Labeling $L_F : V_F \to \Sigma$ is consequently defined, where a symbol `c` in the pattern that matches a node labeled with `c` in the graph will represent the fact a clause *not* satisfied by a certain half-assignment in X is *actually* satisfied by a certain half-assignment in Y. The `d` symbols are sort of "don't care", and `b` and `e` symbols synchronize the whole.

$$L_F(u) = \begin{cases} \texttt{b} & \text{if} \quad u = b \\ \texttt{e} & \text{if} \quad u = e \\ \texttt{c} & \text{if} \quad u = c_{j,h} \\ \texttt{d} & \text{if} \quad u = d_{j,h} \end{cases}$$

As shown in Fig. 1, the edges in the set E_F connect b to every $c_{h,1}$ and $d_{h,1}$, and connect every $c_{h,k}$ and $d_{h,k}$ to e, for $1 \leq h \leq k$. Moreover, there is an edge for every pair of nodes that share the same j and are consecutive in terms of h coordinate (e.g. $c_{j,h}, d_{j,h+1}$), for $1 \leq j \leq 2^{\frac{n}{2}}$ and $1 \leq h \leq k$.

$$\begin{aligned} E_F = &\{(b, c_{j,1}) \mid c_{j,1} \in V\} \cup \{(b, d_{j,1}) \mid d_{j,1} \in V\} \cup \{(c_{j,k}, e) \mid c_{j,k} \in V\} \cup \{(d_{j,k}, e) \mid d_{j,k} \in V\} \\ &\cup \{(c_{j,h}, c_{j,h+1}) \mid c_{j,h}, c_{j,h+1} \in V\} \cup \{(c_{j,h}, d_{j,h+1}) \mid c_{j,h}, d_{j,h+1} \in V\} \\ &\cup \{(d_{j,h}, c_{j,h+1}) \mid d_{j,h}, c_{j,h+1} \in V\} \cup \{(d_{j,h}, d_{j,h+1}) \mid d_{j,h}, d_{j,h+1} \in V\} \end{aligned}$$

We observe that pattern occurrences in G_F have some combinatorial properties.[2]

▶ **Lemma 7.** *If subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ matches in G_F then all the nodes matching P_{x_i} share the same j coordinate and have distinct and consecutive h coordinates (i.e. either $c_{j,h}$ or $d_{j,h}$ for $1 \leq h \leq k$).*

Proof. Gadget G_F contains a single node b with label $L(b) = \mathtt{b}$ and a single node e with label $L(e) = \mathtt{e}$. Morever, the shortest path from b to e contains $k+2$ nodes (b and e included). As $\mathtt{b}P_{x_i}\mathtt{e}$ contains $k+2$ symbols, its matching path $\pi = b, u_1, \ldots, u_k, e$ in G_F must traverse all distinct nodes by construction. Suppose by contradiction that at least one node in π has different j coordinate. This means that two consecutive nodes u_h and u_{h+1} in π have coordinates j and j', with $j \neq j'$. Node u_h is actually either $c_{j,h}$ or $d_{j,h}$, whereas u_{h+1} is either $c_{j',h+1}$ or $d_{j',h+1}$. By inspection of these four possible cases, we observe that our construction of G_F does not provide any edge connecting u_h and u_{h+1}. Indeed, there is no edge that allows a node to change the j coordinate in the middle of a path. Hence we reach a contradiction. Finally, if one of the matching nodes were not consecutive in terms of h coordinate, by construction we know that we would not be following the shortest path to $e_W^{(j)}$ hence it would not be possible to complete the match. ◀

▶ **Lemma 8.** *Subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ matches in G_F if and only if there is $y_j \in Y$ such that the truth assignment $x_i y_j$ satisfies F (i.e. $x_i y_j \models F$).*

Proof. By Lemma 7, we can focus on the k distinct nodes matching P_{x_i}, sharing the same coordinate j. We handle the two implications of the statement individually.

($\Rightarrow$) Consider the partial assignment $x_i \in X$. From the structure of the pattern we know that x_i satisfies all the clauses c_h for which $P_{x_i}[h] = \mathtt{d}$. Since P_{x_i} has a match in G_F, consider the assignment $y_j \in Y$ where j exists by Lemma 7, as observed above. We observe that by construction y_j satisfies those clauses that x_i cannot satisfy, namely those for which $P_{x_i}[h] = \mathtt{c}$. Hence we have found a truth assignment $x_i y_j$ that satisfies F.

($\Leftarrow$) Consider a truth assignment $x_i y_j$ that satisfies F, that is, all clauses c_h for SAT formula F are true. Consider now the nodes with coordinate j in G_F. For $h = 1, 2, \ldots, k$, if $x_i \models c_h$ then $P_{x_i}[h] = \mathtt{d}$ and matching node $d_{j,h}$ exits in G_F by its definition. If $x_i \not\models c_h$ then it must be $y_j \models c_h$: thus $P_{x_i}[h] = \mathtt{c}$ and a matching node $c_{j,h}$ exists in G_F by its construction. The definition of the edges of G_F ensures that all the above nodes $c_{j,h}$ and $d_{j,h}$, as we need, are properly linked to form a path of distinct nodes (for increasing values of h); it is so because they all share the same j coordinate. This implies that P_{x_i} matches in G_F. ◀

While the previous gadget is useful to check whether a half-assignment x_i satisfies F using a *given* subpattern $P_{x_i} \in \mathtt{b}\{\mathtt{c},\mathtt{d}\}^k\mathtt{e}$, we need another "jolly" gadget that matches *all*

[2] Gadget G_F is analogous to the main component of the SETH reduction to regular expression matching of type $|\cdot|$ in [6].

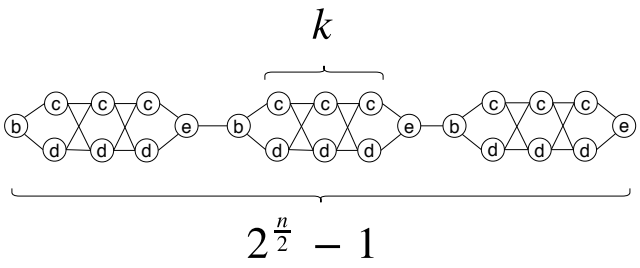

Figure 2 In this example $n = 4, 2^{\frac{n}{2}} = 4$ and $k = 3$. This graph can match any sequence of patterns P_{x_i} but its length is limited to $2^{\frac{n}{2}} - 1$.

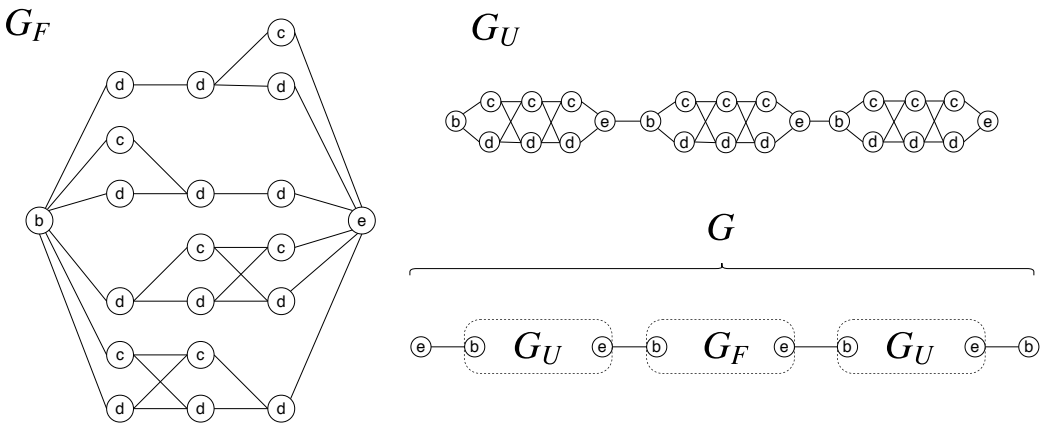

Figure 3 This figure shows how to construct G in the example we proposed before, where $n = 4, 2^{\frac{n}{2}} = 4$ and $k = 3$. What we need to introduce to obtain the final graph G are the two solid edges that are connecting the two instances of G_U with G_F in the bottom part of the figure.

subpatterns in $\mathtt{b}\{\mathtt{c},\mathtt{d}\}^k\mathtt{e}$ (this is useful when x_i does not satisfy F). We concatenate $2^{\frac{n}{2}} - 1$ instances of the latter gadget, thus obtaining the graph $G_U = G(V_U, E_U, L_U)$ illustrated in Figure 2, whose definition is clear from the picture. The jth copy of the gadget substructure has a node b_j followed by nodes $c_{j,h}$, $d_{j,h}$ and then node e_j, with $1 \leq j \leq 2^{\frac{n}{2}} - 1$ and $1 \leq h \leq k$. The labels are $L_U(b_j) = \mathtt{b}$, $L_U(c_{j,h}) = \mathtt{c}$, $L_U(d_{j,h}) = \mathtt{d}$ and $L_U(e_j) = \mathtt{e}$ (we may think about nodes $c_{i,h}$ and $d_{i,h}$ as disposed along two parallel lines). We place the edges $(b_i, c_{i,1})$, $(b_i, d_{i,1})$, $(c_{i,k}, e_j)$, $(d_{i,k}, e_i)$ for connecting the beginning and ending nodes of each gadget with its inner part. We connect nodes $c_{i,h}$ and $d_{i,h}$ with the edges $(c_{i,h}, c_{i,h+1})$, $(c_{i,h}, d_{i,h+1})$, $(d_{i,h}, c_{i,h+1})$, $(d_{i,h}, d_{i,h+1})$. We concatenate our gadgets one after the other using the edges (e_i, b_{i+1}), for $i = 1, \ldots, 2^{\frac{n}{2}} - 1$.

2.3 Putting all together

Armed with gadgets G_F and G_U, we obtain the graph $G = (V, E, L)$ from the SAT formula F by combining them as illustrated in Figure 3. We take one instance of $G_F = (V_F, E_F, L_F)$ and two instances of G_U, say $G_U^{(1)} = (V_U^{(1)}, E_U^{(1)}, L_U^{(1)})$ and $G_U^{(2)} = (V_U^{(2)}, E_U^{(2)}, L_U^{(2)})$, and two new nodes u and z, where their label is respectively $\mathtt{e}$ and $\mathtt{b}$. Then $G = (V, E, L)$ has node set $V = V_F \cup \{u, z\} \cup V_U^{(1)} \cup V_U^{(2)}$, preserving the node labels. The edge set is the union of

the previous edge sets plus four edges: one connects the "last" node labeled with `e` in $G_U^{(1)}$ with the node labeled with `b` in G_F; the other connects the node labeled with `e` in G_F with the "first" node labeled with `b` in $G_U^{(2)}$, plus u is connected to the first node labeled with `b` in $G_U^{(1)}$, and the last node labeled with `e` in $G_U^{(2)}$ is connected to z.

▶ Remark. Each edge connecting a node labeled with `e` to a node labeled with `b` is a *bridge* in G (i.e. its removal disconnect G). As we shall see, the purpose of these bridges is dual since, within a matching path, the ith occurrence of `eb` in the pattern matches the ith bridge with labels `e` and `b` at its endpoints: (i) they synchronize the distinct subpatterns with the distinct (portions of the) gadgets, and (ii) they guarantee that the pattern matches a path of distinct nodes rather than a walk.

We now prove that the reduction is correct, first focusing on subpatterns of P.

▶ **Lemma 9.** *Pattern P matches in G if and only if a subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ of P matches in G_F.*

Proof. For the $\Rightarrow$ implication, the bridges with endpoints labeled with `e` and `b` can only be traversed once in this direction, as P contains the sequence `eb` but does not contain `be`. Moreover, each occurrence of P must begin with one such bridge and end with another such bridge. For this reason each distinct subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ matches a path from either a distinct portion of $G_U^{(l)}$ $(l = 1, 2)$ or G_F. Recall that $G_U^{(1)}$ and $G_U^{(2)}$ can match at most $2^{\frac{n}{2}} - 1$ subpatterns of P each, while P has $2^{\frac{n}{2}}$ of them. Hence one subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ is forced to have a match in G_F in order to have a full match for P.

The $\Leftarrow$ implication is trivial. In fact, if $\mathtt{b}P_{x_i}\mathtt{e}$ has a match in G_F then we can match $\mathtt{b}P_{x_1}\mathtt{e}, \ldots, \mathtt{b}P_{x_{i-1}}\mathtt{e}$ in $G_U^{(1)}$ and $\mathtt{b}P_{x_{i+1}}\mathtt{e}, \ldots, \mathtt{b}P_{x_{2^{\frac{n}{2}}}}\mathtt{e}$ in $G_U^{(2)}$ by construction, and have a full match for P in G. ◀

The main result proves the correctness of our reduction.

▶ **Theorem 10.** *Pattern P matches in G if and only if the* SAT *formula F is satisfiable.*

Proof. By Lemma 9, P matches in G if and only if a subpattern P_{x_i} matches in G_F. By Lemma 8 this holds if and only if the truth assignment $x_i y_j$ satisfies F, hence F is satisfiable. ◀

2.4 Cost of the reduction

We analyze the cost of building the pattern P and the graph G from the SAT formula F.

▶ **Lemma 11.** *Given a* SAT *formula F with n variables, the corresponding pattern P and graph G can be built in $\tilde{O}(2^{\frac{n}{2}})$ time and space.*

Proof. Checking if an assignment satisfies a clause takes $O(n)$ time which, for our goals, is negligible when compared to $\tilde{O}(2^{\frac{n}{2}})$. Recalling that the number k of clauses is polynomially bounded in n, we observe that each P_{x_i} in P has k symbols that can be either `c` or `d` plus symbols `b` and `e`. Since P has $2^{\frac{n}{2}}$ sub-patterns P_{x_i}, summing everything up we get a length of $m = (k+2)\,2^{\frac{n}{2}} = \tilde{O}(2^{\frac{n}{2}})$ symbols. As for G_U, it has $2^{\frac{n}{2}}$ gadgets each one having k nodes labeled with `c`, k nodes labeled with `d`, and nodes b_i and e_i. Hence there are $(2+2k)\,2^{\frac{n}{2}} = \tilde{O}(2^{\frac{n}{2}})$ total nodes. Each node has a constant number of incident edges (at most 4) thus their size is $\tilde{O}(2^{\frac{n}{2}})$ as well. As for G_F, it has $O(k\,2^{\frac{n}{2}})$ nodes labeled with `c` and the same amount of nodes labeled with `d` plus those with `b` and `e`. In this case, each node has a constant number of edges but for `b` and `e`. Nevertheless, `b` and `e` have $O(2^{\frac{n}{2}})$

edges each, therefore the total amount of edges is again $\tilde{O}(2^{\frac{n}{2}})$. For connecting G_F to the two instances of G_U we are adding just 2 edges. Since the pattern and the graph have size $\tilde{O}(2^{\frac{n}{2}})$, we conclude that the cost of our reduction is indeed $\tilde{O}(2^{\frac{n}{2}})$. ◀

2.5 Implications on SETH

The last step in our proof of Theorem 4 is showing that any $O(|E|^{1-\epsilon}\, m)$-time or $O(|E|\, m^{1-\epsilon})$-time algorithm for PMLG unavoidably leads to a failure of SETH. To this aim, assume that we have such an algorithm, say A. Given a SAT formula F we perform our reduction stated in Theorem 10 obtaining pattern P and graph G in $\tilde{O}(2^{\frac{n}{2}})$ time by Lemma 11, observing that $|E| = \tilde{O}(2^{\frac{n}{2}})$ and $m = \tilde{O}(2^{\frac{n}{2}})$. At this point, no matter whether A has $O(|E|^{1-\epsilon}m)$ or $O(|E|\, m^{1-\epsilon})$ time complexity, we will end up with an algorithm deciding if F is satisfiable in $\tilde{O}(2^{\frac{n}{2}}\, 2^{\frac{n}{2}(1-\epsilon)}) = \tilde{O}(2^{\frac{(2-\epsilon)}{2}n})$ time. Since $\alpha = \frac{(2-\epsilon)}{2} < 1$ we conclude that this implies to be able to solve SAT in $O(2^{\alpha n})$ time with $\alpha < 1$, making SETH false.

3 From undirected graphs to DAGs, with binary alphabets

In this section we show that the graph G obtained from the reduction described in Section 2 can be transformed so that each node has degree at most three and label chosen from an alphabet of two symbols $\{\mathtt{0}, \mathtt{1}\}$.

We describe how to modify the proof of Theorem 4 so that it holds for any graph of degree at least three. We observe that the graph built in the reduction in Section 2 has degree $O(2^{\frac{n}{2}})$. To obtain degree at most three, we first modify gadgets G_F and G_U to meet such requirement, and then adjust pattern P consequently. Finally, we obtain a binary alphabet for the labels, thus proving Corollary 5.

After that we prove Corollary 6, showing that the undirected graph can be easily transformed into a directed acyclic graph (DAG).

3.1 Maximum degree three

Revised gadget G_F

As depicted in Figure 4*(a)*, consider the $O(2^{\frac{n}{2}})$ edges connecting node b with nodes $c_{j,1}$ and $d_{j,1}$ in G_F. We replace them by a binary tree structure whose nodes are new dummy nodes f_l with labels $L(f_l) = \mathtt{d}$ for $1 \leq l \leq 2^{\frac{n}{2}} - 2$. As for node e, we proceed along the same way and replace the edges connecting nodes $c_{j,k}$ and $d_{j,k}$ to e by a binary tree structure (this case is not shown in the figure). The internal nodes of these trees have degree at most three.

This is not enough to guarantee degree at most three for each node in G_F as nodes $c_{j,h}$ and $d_{j,h}$ could have degree four. For example, with some nodes $d_{j,h-1}, d_{j,h}$ and $d_{j,h+1}$, nodes $c_{j,h-1}, c_{j,h}$ and $c_{j,h+1}$ could exists. Then both $c_{j,h}$ and $d_{j,h}$ would have degree four. This can be fixed as shown in Figure 4*(b)*, adding two pairs of dummy nodes f with label $L(f) = \mathtt{d}$ to lower the degree to three.[3]

At this point, we added $O(2^{\frac{n}{2}})$ dummy nodes f for the binary tree, and $O((k-1)2^{\frac{n}{2}}) = \tilde{O}(2^{\frac{n}{2}})$ pairs of nodes $f^{(1)}_{j,h}, f^{(2)}_{j,h}$. Moreover, the new edges for the binary tree are as many as

[3] One pair is placed between nodes $c_{j,h-1}, d_{j,h-1}$ and nodes $c_{j,h}, d_{j,h}$ via edges $(c_{j,h-1}, f^{(1)}_{j,h-1}), (d_{j,h-1}, f^{(1)}_{j,h-1})$ and $(f^{(2)}_{j,h-1}, c_{j,h}), (f^{(2)}_{j,h-1}, d_{j,h})$. The other pair of dummy nodes f is placed between nodes $c_{j,h}, d_{j,h}$ and nodes $c_{j,h+1}, d_{j,h+1}$ via edges $(c_{j,h}, f^{(1)}_{j,h}), (d_{j,h}, f^{(1)}_{j,h})$ and $(f^{(2)}_{j,h}, c_{j,h+1}), (f^{(2)}_{j,h}, d_{j,h+1})$.

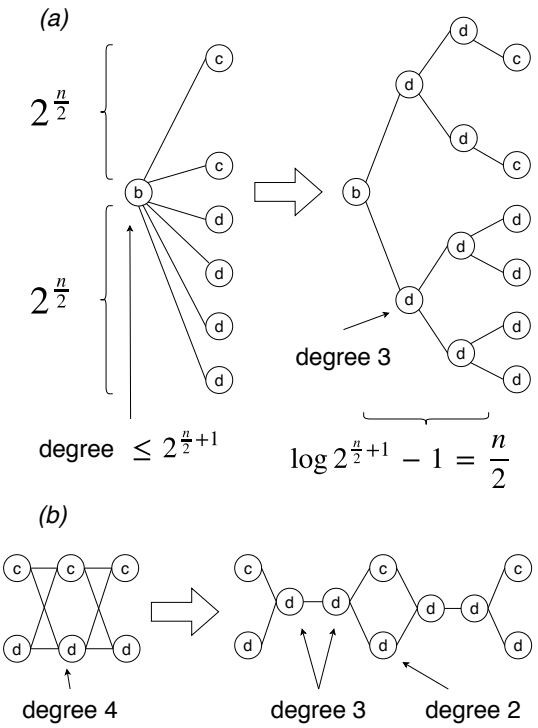

Figure 4 The transformation of gadget G_F. *(a)* The edges connecting node b with nodes $c_{j,1}$ and $d_{j,1}$ are replaced by a binary tree structure. Note that, even if not reported in this figure, the same process is performed for the edges connecting nodes $c_{j,k}$ and $d_{j,k}$ with node e. *(b)* When $c_{j,h-1}, c_{j,h}$ and $c_{j,h+1}$ exist, $c_{j,h}$ and $d_{j,h}$ have degree four. Hence pairs of dummy nodes are added to achieve degree three.

the nodes while for the other modifications we add one edge for each pair of dummy nodes. The overall time complexity to build the transformed G_F does not increase significantly.

Revised gadget G_U

Gadget G_U has to be consistent with G_F. We add $(\log 2^{\frac{n}{2}+1}) - 1 = \frac{n}{2}$ dummy nodes f with label $L(f) = \texttt{d}$ between every b_i node and the nodes $c_{i,1}$ and $d_{i,1}$ following it. We also add $\frac{n}{2}$ dummy nodes f with label $L(f) = \texttt{d}$ between every node e_i and the previous nodes $c_{i,k}$ and $d_{i,k}$. We are adding $2\frac{n}{2}(2^{\frac{n}{2}} - 1) = \tilde{O}(2^{\frac{n}{2}})$ new nodes and one new edge per node, thus the overall time complexity will not be affected. The need for this step will be clearer when we will modify pattern P, as it has to match either G_F or G_U, so the same format of P is required in both types of gadgets.

We have another issue to handle. As in G_F, there could be nodes $c_{i,h}$ and $d_{i,h}$ of degree four. In that case, we add pairs of dummy nodes f with label $L(f) = \texttt{d}$ following the same schema presented for G_F and illustrated in Figure 4*(b)*. In this way we are introducing $2(k-1)(2^{\frac{n}{2}} - 1) = \tilde{O}(2^{\frac{n}{2}})$ new nodes and one edge for each pair of dummy nodes which do not change the time complexity of the reduction.

Revised pattern P

Pattern $P = \mathtt{eb}P_{x_1}\mathtt{eb}P_{x_2}\mathtt{e}\dots\mathtt{b}P_{x_{2^{\frac{n}{2}}}}\mathtt{eb}$ is modified so as to match G_F and G_U when needed. We add $\frac{n}{2}$ symbols $\mathtt{d}$ after each occurrence of $\mathtt{b}$ and before each occurrence of $\mathtt{e}$. Moreover, we insert $\mathtt{d}$ symbols inside the subpatterns $P_{x_i} = a_1\, a_2 \dots a_k$, where $a_h \in \{\mathtt{c},\mathtt{d}\}$, to obtain the new subpatterns $P'_{x_i} = a_1\mathtt{d}\,\mathtt{d}\,a_2\,\mathtt{d}\,\mathtt{d}\dots\mathtt{d}\,\mathtt{d}\,a_k$. Therefore, the new pattern to match will be

$$P' = \mathtt{eb}\underbrace{\mathtt{d}\dots\mathtt{d}}_{\frac{n}{2}\text{ times}}P'_{x_1}\underbrace{\mathtt{d}\dots\mathtt{d}}_{\frac{n}{2}\text{ times}}\mathtt{e}\dots\mathtt{b}\underbrace{\mathtt{d}\dots\mathtt{d}}_{\frac{n}{2}\text{ times}}P'_{x_{2^{\frac{n}{2}}}}\underbrace{\mathtt{d}\dots\mathtt{d}}_{\frac{n}{2}\text{ times}}\mathtt{eb}.$$

It is worth noting that $P' \in \mathtt{eb}\,(\{\mathtt{c},\mathtt{d}\}^+\mathtt{eb})^+$ in this way. The number of new symbols added before and after the subpatterns is $\frac{n}{2}2^{\frac{n}{2}} = \tilde{O}(2^{\frac{n}{2}})$ while the ones inserted inside them are $2(k-1)2^{\frac{n}{2}} = \tilde{O}(2^{\frac{n}{2}})$. The time cost of the reduction does not increase significantly.

3.2 Binary alphabet

The last step consists in defining a binary encoding α of the symbols $\Sigma = \{\mathtt{b},\mathtt{e},\mathtt{c},\mathtt{d}\}$, namely,

$$\alpha(\mathtt{c}) = \mathtt{0000},\quad \alpha(\mathtt{d}) = \mathtt{1111},\quad \alpha(\mathtt{b}) = \mathtt{10},\quad \alpha(\mathtt{e}) = \mathtt{01}.$$

Given any string $x = x[1..m]$, we define its binary encoding $\alpha(x) = \alpha(x[1])\cdots\alpha(x[m])$. The following useful synchronizing property holds, recalling that each edge connecting a node with label $\mathtt{e}$ to a node with label $\mathtt{b}$ is a bridge in (transformed) G.

▶ **Lemma 12.** *For any string $x \in \Sigma^+$, its binary encoding $\alpha(x)$ contains* $\mathtt{0110}$ *if and only if x contains* $\mathtt{eb}$.

Proof. We observe that $\mathtt{e}$ and $\mathtt{b}$ are encoded by two bits each, while $\mathtt{c}$ and $\mathtt{d}$ are enconed by four bits each. Hence, $\mathtt{0110}$ can appear by concatenating the binary encoding of two or three symbols. On the other hand, $\mathtt{eb}$ occurs in x if and only if it occurs in a substring of length 3 of x. Consequently, it suffices to check the claim by inspection of all the 64 substrings of x of length 3, $\mathtt{ccc}$, …, $\mathtt{eee}$, and their encodings to see that the property holds. ◀

Any walk matched by the revised pattern P' crosses the bridges in the direction from $\mathtt{e}$ to $\mathtt{b}$.

▶ **Lemma 13.** *For any pattern P' obtained in the reduction, its binary encoding $\alpha(P')$ does not contain* $\mathtt{1001} = \alpha(\mathtt{be})$.

Proof. Recalling that $P' \in \mathtt{eb}\,(\{\mathtt{c},\mathtt{d}\}^+\mathtt{eb})^+$, all the possible substrings of length 3 in P' by construction are of the forms $\mathtt{b}\{\mathtt{c},\mathtt{d}\}\mathtt{e}$, $\mathtt{eb}\ \{\mathtt{c},\mathtt{d}\}$, $\{\mathtt{c},\mathtt{d}\}\mathtt{eb}$, $\mathtt{b}\{\mathtt{c},\mathtt{d}\}^2$, $\{\mathtt{c},\mathtt{d}\}^2\mathtt{e}$, and $\{\mathtt{c},\mathtt{d}\}^3$. By inspection of this small number of cases, none contains $\mathtt{be}$, and none of their binary encodings contains $\mathtt{1001}$. ◀

An immediate consequence of Lemma 12 and 13 is that the encodings preserve the occurrences. Let G' be the transformed graph, and P' be the revised pattern in the reduction. Let $\alpha(G')$ denote the graph obtained from G' by relabeling its nodes with the binary encoding α of their labels.

▶ **Lemma 14.** *In the reduction, P' matches in G' if and only if $\alpha(P')$ matches in $\alpha(G')$.*

Proof. It follows by Lemma 12 and 13, and the fact that all the edges whose endpoints have one label $\mathtt{e}$ and the other label $\mathtt{b}$ are bridges, and they are traversed in the direction from $\mathtt{e}$ to $\mathtt{b}$ when matching P'. ◀

In the encoding above, each node stores two or four bits. By replacing it with a chain of two or four nodes with a single bit as a label, we obtain the proof of Corollary 5.

3.3 Directed acyclic graphs

In order to prove Corollary 6, we observe that the proof of Theorem 4 can be easily modified in order to work also for DAGs.

Considering the definitions of edges E_F and E_U in the proof of Theorem 4, and the transformation described so far, we immediately obtain a directed graph G' that is acyclic. Indeed, because of bridges and occurrences of `eb` in the pattern, each pattern match must begin with some bridge, end with a different bridge and lay along a path from the first to the last bridge in the graph. So the edges can be oriented by construction from left (first bridge) to right (last bridge), as it can be checked in Fig. 1–4.

4 Conclusions

We studied the complexity of pattern matching on labeled graphs, giving a SETH conditional quadratic lower bound for the exact pattern matching. In strings the exact pattern matching takes linear time whereas the approximate pattern matching takes quadratic time under a matching conditional lower bound. Differently from strings, our result along with the upper bounds in [3, 24] imply that the exact and approximate pattern matching (the latter with errors in the pattern) have the same complexity under the SETH conjecture. Our conditional lower bound uses a binary alphabet and holds even if restricted to nodes of maximum degree at most three for undirected graphs, and to nodes of maximum sum of indegree and outdegree at most three for directed acyclic graphs (DAGs).

Two border cases are left if the maximum degree or sum of indegree and outdegree is at most two: a) when the undirected graph is a simple path or a cycle, and pattern matching goes along a walk (so it is a sort of zig-zag string matching), and b) when the graph is a directed cycle. For a), we can convert each edge into a pair of arcs and apply the known quadratic algorithm in [3]. On the other hand, we can extend our reduction to derive a matching SETH lower bound for this case [12]. For b), we can adapt any known string matching algorithm (e.g. [19]) to get linear time.

An interesting and natural question for directed graphs is what happens when the graph is deterministic, that is, for each symbol c and each node v, there is at most one neighbor of v labeled with c. Unfortunately, this does not make the problem any easier. Although our reduction creates an inherently non-deterministic graph, it is possible to alter the reduction scheme to create a deterministic graph [12].

Acknowledgements

The first two authors are grateful to Alessio Conte and Luca Versari for providing their comments on the reduction. The last author wishes to thank the participants of the annual research group retreat on sparking the idea to study SETH reductions in this context. We thank the anonymous reviewers of an earlier version of this paper for useful suggestions for improving the presentation and for pointing out the connection to regular expression matching.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.

2 Tatsuya Akutsu. A linear time pattern matching algorithm between a string and a tree. In *4th Symposium on Combinatorial Pattern Matching, Padova, Italy*, pages 1–10, 1993.

3 Amihood Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern matching in hypertext. *J. Algorithms*, 35(1):82–99, 2000.

4 Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM.

6 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016.

7 Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70, pages 311–321. PMLR, 2017.

8 Karl Bringmann and Marvin Kunnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, pages 79–97, Washington, DC, USA, 2015. IEEE Computer Society.

9 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018.

10 Alessio Conte, Gaspare Ferraro, Roberto Grossi, Andrea Marino, Kunihiko Sadakane, and Takeaki Uno. Node similarity with q -grams for real-world labeled networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1282–1291, 2018.

11 Massimo Equi. Pattern matching in labeled graphs. Master's thesis, University of Pisa, Italy, 2018.

12 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of exact pattern matching in graphs: Determinism and zig-zag matching. Manuscript in preparation, 2019.

13 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1433–1445, 2018.

14 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017.

15 Garrison Erik, Sirén Jouni, Novak Adam M, Hickey Glenn, Eizenga Jordan M, Dawson Eric T, Jones William, Garg Shilpa, Markello Charles, Lin Michael F, Paten Benedict, and Durbin Richard. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36:875, aug 2018.

16 Shohei Hido and Hisashi Kashima. A linear-time graph kernel. In Wei Wang 0010, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu, editors, *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, pages 179–188. IEEE Computer Society, 2009.

17 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367 – 375, 2001.

18 Chirag Jain, Haowen Zhang, Yu Gao, and Srinivas Aluru. On the Complexity of Sequence to Graph Alignment. *bioRxiv*, 2019.

19 Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350, March 1977.

20 U. Manber and S. Wu. Approximate string matching with arbitrary costs for text and hypertext. In *IAPR Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland*, pages 22–33, 1992.

21 Gonzalo Navarro. Improved approximate pattern matching on hypertext. *Theoretical Computer Science*, 237(1-2):455–463, 2000.

22 K. Park and D. Kim. String matching in hypertext. In *6th Symposium on Combinatorial Pattern Matching, Espoo, Finland*, page 318, 1995.

23 Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115, January 2008.

24 Mikko Rautiainen and Tobias Marschall. Aligning sequences to general graphs in $O(V+mE)$ time. *bioRxiv*, pages 216–127, 2017.

25 Marko A. Rodriguez. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages, Pittsburgh, PA, USA, October 25-30, 2015*, pages 1–10, 2015.

26 Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, Norman Warthmann, Sandra Gesing, Oliver Kohlbacher, and Detlef Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome Biology*, 10:R98, 2009.

27 Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and Philip S. Yu. A survey of heterogeneous information network analysis. *IEEE Trans. Knowl. Data Eng.*, 29(1):17–37, 2017.

28 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 11(2):375–388, March 2014.

29 Kavya Vaddadi, Naveen Sivadasan, Kshitij Tayal, and Rajgopal Srinivasan. Sequence alignment on directed graphs. *bioRxiv*, 2017.

30 Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 745–754, 2012.